CS 4341/6341: Team Project

A cool and useful way to learn is to apply ideas from a course into something of your own making. In working through a creative application of core principles, one gets to see the principles at work, which becomes a basis by which these same principles could be applied in the future. Another effective way to learn is to engage in discussion and social activity that's focused on learning. That is, to learn by learning what others are learning. In this project, we will combine both these types of learning: (1) learn by applying principles in this course to something new; (2) learn from others.

In this project, you will search for an application of some continuous math that's not already in the course and produce the following:

- A written description and explanation of the math (with intuition).
- Pseudocode for a computational version of the same math, along with an explanation.
- A simplified "bare bones" example of the computation that gets to the essence of how it works, that aligns with the pseudocode, and which visually demonstrates the application.
- A second computational example with two parts:
    1. An exercise for other students to do that involves filling in some code, as you've done in numerous module exercises.
    2. A solution to that exercise.
- A write-up of the exercise (intended for other students).
- A description of who-did-what in your team.
- Then, later, you will do other a couple of exercises that other teams produce (so that you learn what others are learning).

Thus, to sum up: you will create an application and an exercise for others to learn your application of a core principle. You will need to think about this carefully, because we someone else in the class will try out your exercise and we will see if they actually learned. (By symmetry, you will be doing someone else's project exercises.)

An example: Let's examine how this would work using, say, an example from Module 3:

- First, take a quick look at the last section of Module 3, to remind yourself.
- Suppose the core math principle is to explain how Euler integration works.
- The last section of Module 3 shows an example first in math:

$$\begin{aligned} v(t+s) &= v(t) + s\ a(t) \\ d(t+s) &= d(t) + s\ v(t) \end{aligned}$$

  This would be accompanied with an explanation.
- Then comes the computational version:

```
t = 0;
v = 0;
d = 0;
while ( ... )
    t = t + s;
    v = v + s * a;
```

```
            d = d + s * v;
        }
```

Again, this would be accompanied with explanation.
- Next is the actual implementation, which is the code and simple GUI in movingobject.zip.
- Exercise 40 then has the "exercise".

Your team's goals:

- Find a topic of continuous math that we have not covered in the class, but that also avoids linear algebra. Linear algebra is continuous math, but is too specialized and does not fit into the themes of our course.
- Write a short justification (2-3 sentences) of why this topic fits into the category of *continuous math*, and why this is sufficiently distinct from the topics in class.
- Explain this topic and associated equations, using pictures if necessary.
- Write out a computational version (pseudocode) of the same that you can explain.
- Develop a computational implementation with a visual component so something can be "seen".
- Develop a simplified version (perhaps without a GUI) as an exercise that explains the core math idea or principle. The exercise should have template code, instructions in PDF, and a solution. The code that needs to be filled in should be relatively short, do-able in 15-20 minutes.
- NOTE: your code will have to be fully self-contained and written in Java. You cannot use an API or library other than what's already standard in the Java API. And your examples need to be at the most basic level. That is, you cannot implement cryptic looking methods with borrowed code that do most of the real math, while your code merely uses it.
- You need to include instructions for compiling and executing at the top of the relevant Java file (one for your application/demo, and one for the exercise).

How the project will be graded:

- 10% for the choice of topic and justification. The topic should be different enough from anything in the modules, yet uses continuous math and relates to the course.
- 25% for the quality of the math explanation (feel free to add pictures, figures etc).
- 25% for the computational example, pseudocode, implementaiton, and accompanying explanation.
- 10% for properly distributed and documented teamwork.
- 15% rating of the exercise from the grader
- 15% rating from two other teams that will be assigned to do your exercise.

So, what kinds of topics might you consider?

- One way to think about this: find a math topic that can be explained via code.
- Another way is to start with applications and work backwards. For example, you could look under the topic of "mathematical models of ..." and see what you find. Another broad category to search for is "computational science".
- Yet another way is to first find an algorithm and work towards an application.

- There are entire shelves of books in the library on the the topic of applied math and mathematical modeling. See, for example, the undergraduate textbook "Introduction to Computational Science: Modeling and Simulation for the Sciences", by Shiflet et al.
- What is NOT allowed is a tiny modification of some concept that's already in the materials. For example, another example similar to the motion example above or to the rabbit-lynx equations.
- Topics we have not explored in this course include: partial differential equations, finite-element method, Markov chains, fractals, optimization techniques.

What to submit:

- Meet as a team and get started with your project well before the *project selection* deadline.
- Submit your project selection in Blackboard. This is a half-page description of your project and your plan for implementation, and what your exercise is likely to be.
- For the main submission, write the idea description and pseudocode in a PDF called `MainIdea.pdf`. This PDF can include helpful pictures to explain the idea/concept. Remember, your grade for the project will depend on someone else succeeding in understanding your project, and completing the exercise.
- Write the exercise directions in a PDF called `Exercise.pdf`.
- Describe who-did-what in your team in `Teamwork.pdf`.
- Your demo should be called `Demo.java`.
- The template code for the exercise should be called `Exercise.java`.
- The solution to the exercise should be called `ExerciseSolution.java`. An explanation of the solution should be in `ExerciseSolution.pdf`.
- Pick a team name. We'll use "hippo" as our example team name below.
- To submit your project itself, upload the entire package as a zip file with three sub-folders:
  1. The primary folder called `team-x-project` where x is your team-name. Thus, for example, if your team name is "hippo", this folder would be called `team-hippo-project`. This folder will have: `MainIdea.pdf` and `Demo.java`.
  2. A second folder called `team-x-exercise`. This will contain: `Exercise.pdf` and `Exercise.java`. Make sure the explanation is clear enough to follow.
  3. A third folder called `team-x-private`. This will contain: `ExerciseSolution.java`, `ExerciseSolution.pdf` (which explains the why the code in `ExerciseSolution.java` is the solution), and `Teamwork.pdf`.
- Upload the zip file to Blackboard under "team project".
- Make sure that the primary and secondary folders (`team-x-project` and `team-x-exercise`) do not have your actual individual names. That is, nobody else should be able to identify who is in team x.
- Keep in mind that the primary and secondary folders (`team-x-project` and `team-x-exercise`) be distributed to other teams. (The folder `team-x-private` will be seen only by the instructional staff.) Make sure that you have enough in your primary folder for the material to be understood and for the exercise to be do-able without having to look at websites.